```
int KeyGenerator::Locations(BitLocations& OrderedPairs, unsigned int CFB_Size)
{   short BitStatus[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};      // create empty array for keeping track of bit status.
    short PossibleChoices[16];                                  // create empty array for keeping track of possible choices.
    short AvailableBits;                                        // variable that will maintain a count of available bits.
    short AssignedBit;                                          // variable that will contain the index of the current assigned bit.
    short FindBit;                                              // index variable used for finding the assigned bit's position.
    unsigned char crc_mrk=MRK.CRC();                            // obtain Master Recovery Key's CRC.
    for(short BitPointer=0;BitPointer<16;BitPointer++)          // loop through 16 bit pointers:
    {   for(AssignedBit=0,AvailableBits=0;AssignedBit<16;AssignedBit++)   // loop through useable bits:
        {   if(BitStatus[AssignedBit]>-1)                       // if this bit is not already in use,
            {   PossibleChoices[AvailableBits]=AssignedBit;     // add it to the list of possibilities.
                AvailableBits++;                                // increment the number of available bits.
            }
        }
        AssignedBit=MRK.Value.c[(crc_mrk+BitPointer)%160]%AvailableBits;  // pick a pseudo-random number between 0 and AvailableBits.
        for(FindBit=0; FindBit<(AvailableBits+1); FindBit++)    // loop through possible locations of the AssignedBit:
        {   if(FindBit==AssignedBit)                            // if FindBit = AssignedBit,
                OrderedPairs.Bit[BitPointer]=PossibleChoices[j];  // PossibleChoices[FindBit] is the pseudo-random bit-location.
        }
        BitStatus[OrderedPairs.Bit[BitPointer]]=-1;             // mark bit as used so that it is not re-used.
        OrderedPairs.Word[BitPointer]=                          // pick a pseudo-random number between 0 and the number of WORDS (16-bit sets)
            (MRK.Value.c[(crc_mrk+BitPointer)%160]%(CFB_Size/2))*2;  // in the current ciphertext block and use it as this WORD pointer.
    }
    return 1;
```

Table 1 Generation of the ECD Code and Insertion Into Ciphertext Block